Type-safe Web APIs with Protocol Buffers in Swift


Hi, I'm Yusuke
@kitasuke

I'm going to talk about...

→ What Protocol Buffers are

I'm going to talk about...

→ What Protocol Buffers are

→ Pros and Cons

I'm going to talk about...

→ What Protocol Buffers are

→ Pros and Cons

→ Where to start

Web APIs


JSON

HTTP Request

JSON → Data

```swift
let json = ["id": 1]
let data = try? JSONSerialization.data(
    withJSONObject: json,
    options: []
)

var request = URLRequest(url: url)
request.httpBody = data
```

HTTP Response

Data → JSON

```swift
// {"user": {"name": "Yusuke"}}
URLSession.shared.dataTask(with: request) { (data, _, _) in
    let json = (try? JSONSerialization.jsonObject(
        with: data!,
        options: []
    )) as? [String: Any]
    let user = json?["user"] as? [String: Any]
    let name = user?["name"] as? String
}
```

{"isTyped": false, "status": "disappointed"}

```
[AnyHashable: Any]
```

Any?

Protocol Buffers

a.k.a protobuf

Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler.

— Google

Serialization format - think JSON, but smaller, faster and safer

— Yusuke

Serialization Format

Why are Protocol Buffers a big deal?

HTTP Request

JSON → Data

```swift
let json = ["id": 1]
let data = try? JSONSerialization.data(
    withJSONObject: json,
    options: []
)

var request = URLRequest(url: url)
request.httpBody = data
```

HTTP Request

~~JSON~~ → Data

HTTP Request

UserRequest ➡ Data

```swift
// custom type
let userRequest = UserRequest.with {
    $0.id = 1
}
let body = try? userRequest.serializedData()

var request = URLRequest(url: url)
request.httpBody = body
```

HTTP Response

Data → JSON

```swift
// {"user": {"name": "Yusuke"}}
URLSession.shared.dataTask(with: request) { (data, _, _) in
    let json = (try? JSONSerialization.jsonObject(
        with: data!,
        options: []
    )) as? [String: Any]
    let user = json?["user"] as? [String: Any]
    let name = user?["name"] as? String
}
```

HTTP Response

Data → JSON

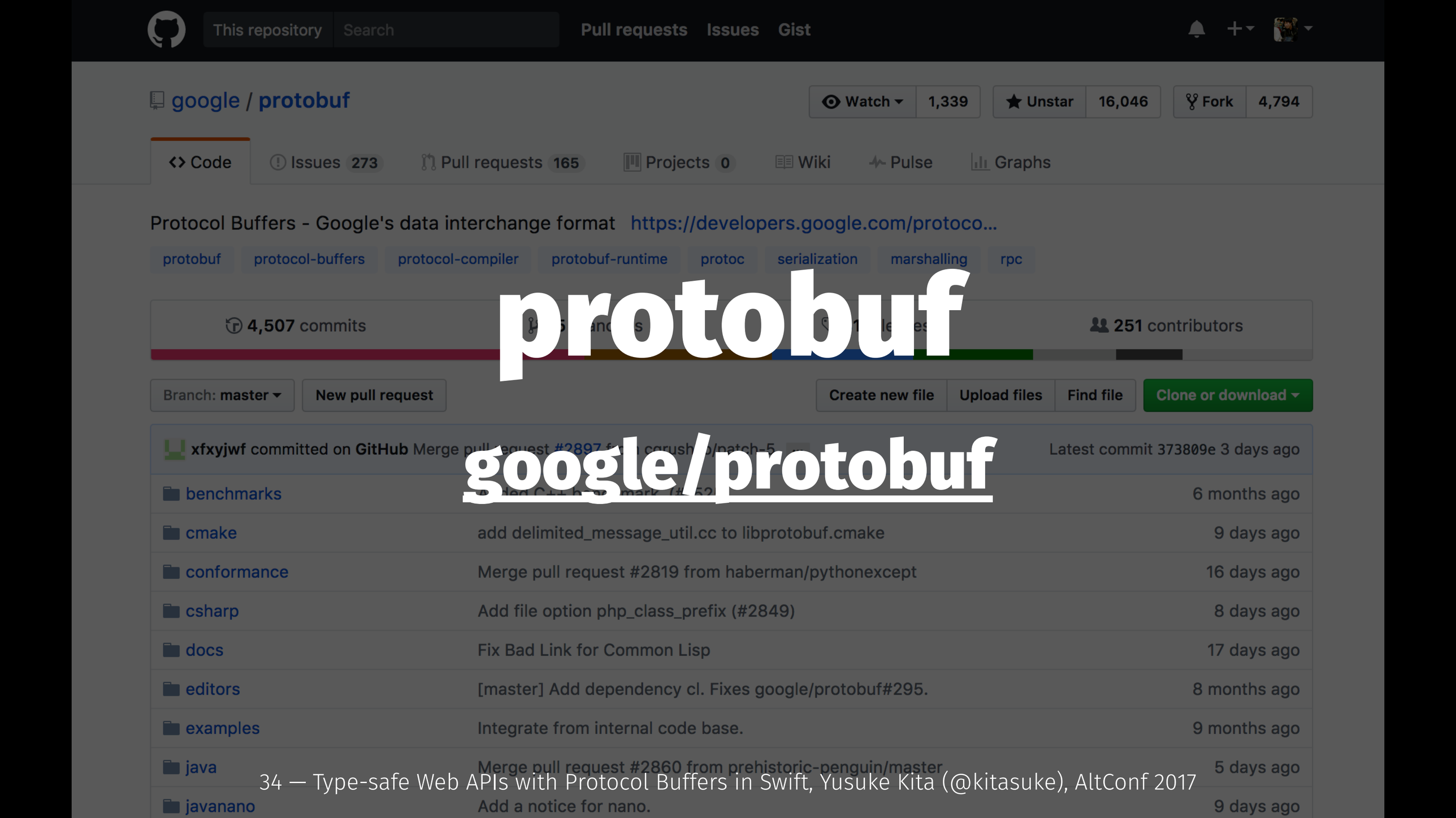
HTTP Response

Data ➡ UserResponse

```swift
// UserResponse(user: User(name: "Yusuke"))
URLSession.shared.dataTask(with: request) { (data, _, _) in
    // custom type
    guard let response = try? UserResponse(serializedData: data!) else {
        // error
        return
    }
    let user = response.user
    let name =  user.name
}
```

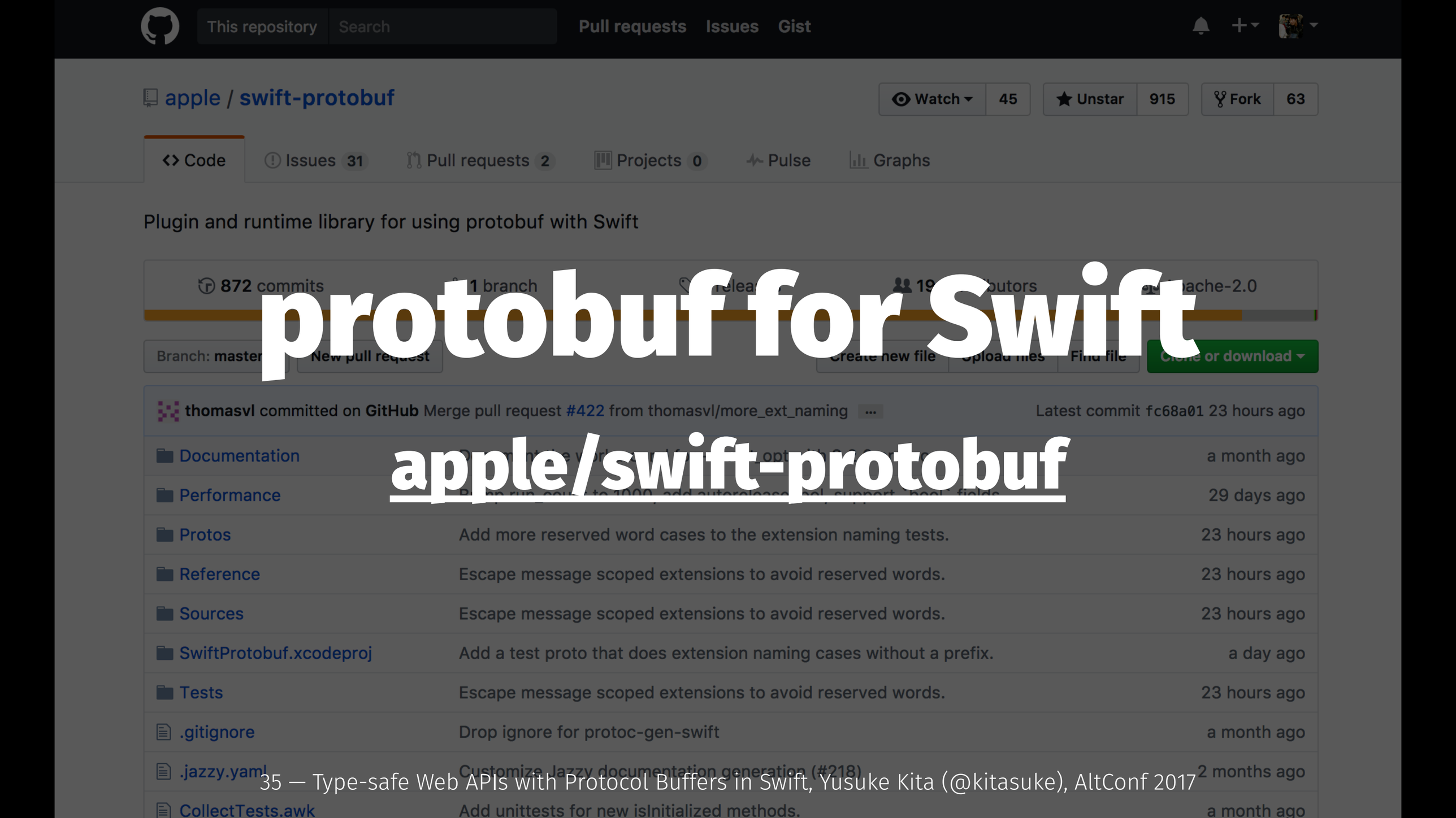
```
Protobuf(
    isTyped: true,
    status: .happy
)
```

But how do they work?



protobuf

google/protobuf



protobuf for Swift

apple/swift-protobuf

"But we still use Objective-C 😢"

1. Define message types
2. Generate code
3. Serialize/Deserialize

1. **Define message types**
2. Generate code
3. Serialize/Deserialize

Message types define data structures in `.proto` files

Message types have key-value pairs

Type-safe API call with Protocol Buffers in Swift

Yusuke Kita

Apple recently open sourced swift-protobuf which is a plugin of Protocol Buffers for swift language. Protocol Buffers enables us to have type safety, make API faster and unify schema of structured data. JSON used to be a reasonable way for API call in most cases, but Protocol Buffers could be another option if we consider these benefits.

In this talk, you'll discover examples of usage with swift-protobuf in server and client apps. Yusuke will also highlight pros and cons compare to JSON based on his knowledge and experiences.

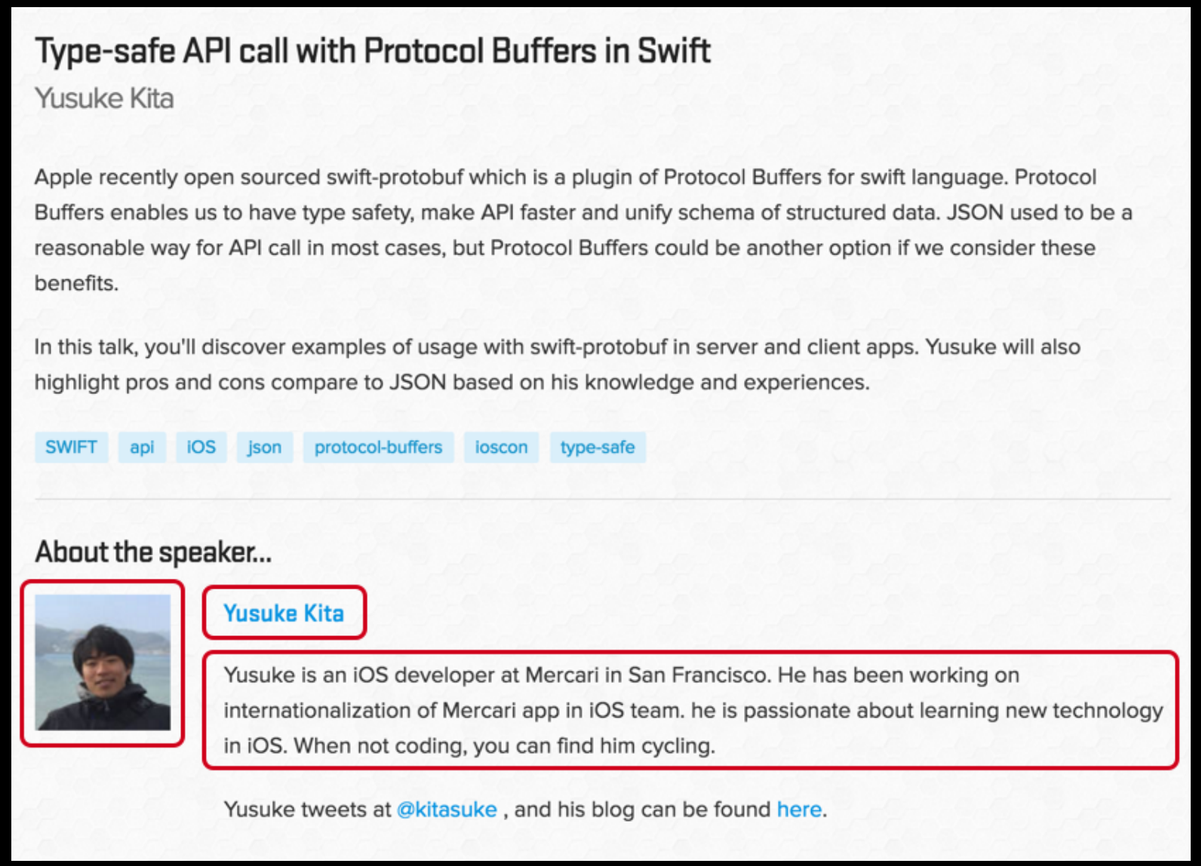
SWIFT api iOS json protocol-buffers ioscon type-safe

About the speaker...

Yusuke Kita

Yusuke is an iOS developer at Mercari in San Francisco. He has been working on internationalization of Mercari app in iOS team. he is passionate about learning new technology in iOS. When not coding, you can find him cycling.

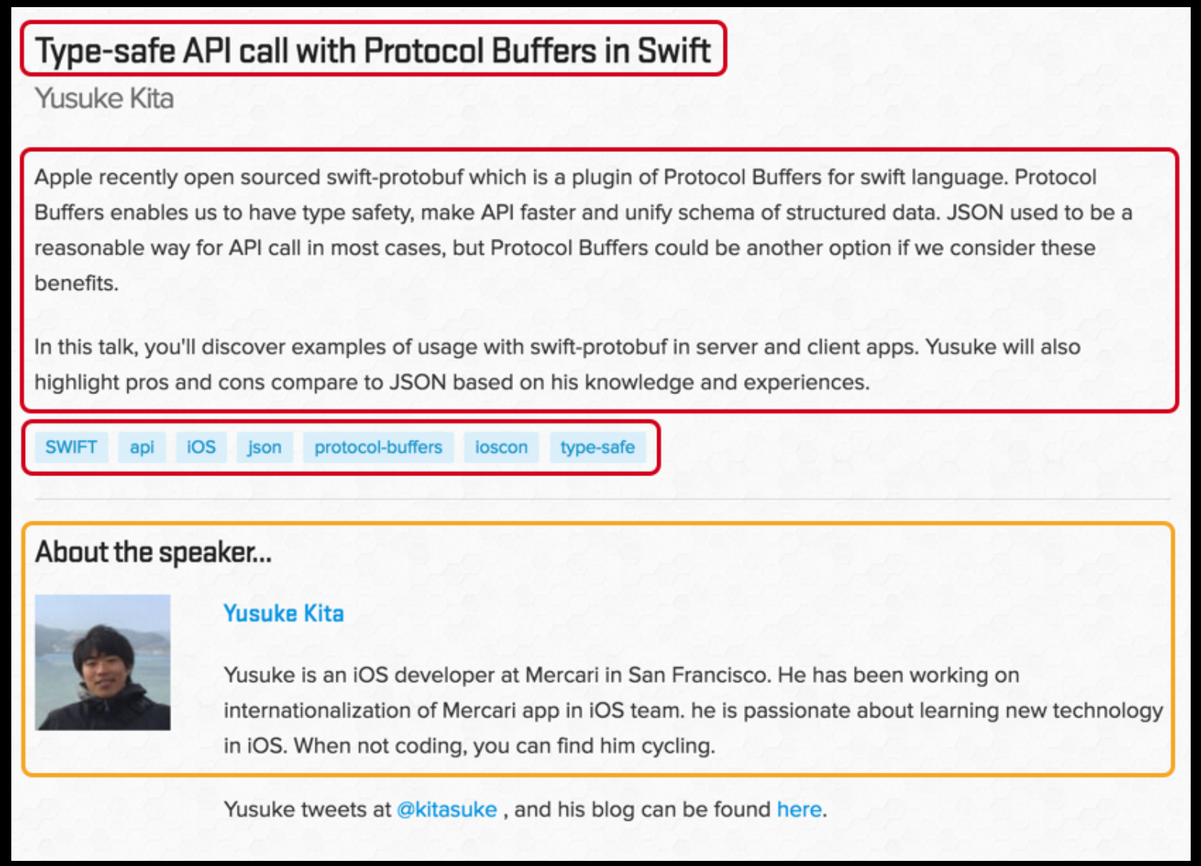
Yusuke tweets at @kitasuke , and his blog can be found here.



user.proto

```
syntax = "proto3"; // protoc version

message User {
    int32 id = 1; // field number
    string name = 2;
    string introduction = 3;
    string photoUrl = 4;
    Type type = 5;

    enum Type {
        Speaker = 0;
        Attendee = 1;
    }
}
```

42 — Type-safe Web APIs with Protocol Buffers in Swift,
Yusuke Kita (@kitasuke), AltConf 2017



talk.proto

```
syntax = "proto3";

import "user.proto";

message Talk {
    int32 id = 1;
    string title = 2;
    string desc = 3;
    User speaker = 4;
    repeated string tags = 5; // Array
}
```

1. Define message types
2. **Generate code**
3. Serialize/Deserialize

protobuf compiler generates code from `.proto` file

Basic types

`Int32`, `UInt32`, `Int64`, `UInt64`, `Bool`, `Float`, `Double`, `String`, `Array`, `Dictionary`, `Data`

Supported languages

C, C++, C#, Go, Haskell, Java, Javascript, **Objective-C**, PHP, Python, Ruby, Rust, Scala, **Swift** etc.

Swift features

→ `struct`, not `class`

→ `enum RawValue` is `Int`

→ Default value is set

user.proto

```
syntax = "proto3"; // protoc version

message User {
    int32 id = 1; // field number
    string name = 2;
    string introduction = 3;
    string photoUrl = 4;
    Type type = 5;

    enum Type {
        Speaker = 0;
        Attendee = 1;
    }
}
```

user.pb.swift

```swift
// struct
struct User: SwiftProtobuf.Message, ... {
    init() {}

    enum Type: SwiftProtobuf.Enum {
        typealias RawValue = Int // always Int
        case speaker // = 0
        case attendee // = 1
        case UNRECOGNIZED(Int)
    }

    // property has default value
    var id: Int32 = 0
    var name: String = String()
    var introduction: String = String()
    var photoURL: String = String()
    var type: User.TypeEnum = .speaker
}
```

talk.proto

```
syntax = "proto3";

import "user.proto";

message Talk {
    int32 id = 1;
    string title = 2;
    string desc = 3;
    User speaker = 4;
    repeated string tags = 5; // Array
}
```

talk.pb.swift

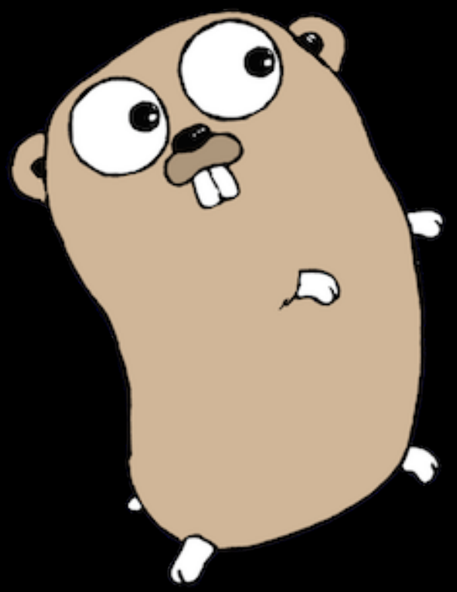
```
struct Talk: SwiftProtobuf.Message, ... {
    init() {}

    var id: Int32 = 0
    var title: String = String()
    var speaker: User = User()
    var desc: String = String()
    var tags: [String] = []
}
```

User.java

```
public static final class User extends Message<User, User.Builder> {
    public final Integer id;
    public final String name;
    public final String introduction;
    public final String photoUrl;
    public final Type type;
}
```

Talk.java

```
public final class Talk extends Message<Talk, Talk.Builder> {
    public final Integer id;
    public final String title;
    public final User speaker;
    public final String summary;
    public final List<String> tags;
}
```



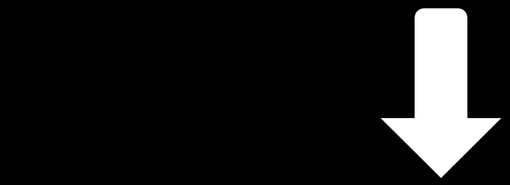
user.pb.go

```
type User_Type int32

const (
    User_Speaker User_Type = 0
    User_Attendee User_Type = 1
)

type User struct {
    ID           int32  `protobuf:"varint,1,opt,name=id,proto3" json:"id,omitempty"`
    Name         string `protobuf:"bytes,2,opt,name=name,proto3" json:"name,omitempty"`
    Introduction string `protobuf:"bytes,3,opt,name=introduction,proto3" json:"introduction,omitempty"`
    PhotoURL     string `protobuf:"bytes,4,opt,name=photoUrl,proto3" json:"photoUrl,omitempty"`
    Type      User_Type `protobuf:"varint,5,opt,name=type,proto3,enum=api.User_Type" json:"type,omitempty"`
}
```

talk.pb.go

```
type Talk struct {
    ID      int32     `protobuf:"varint,1,opt,name=id,proto3" json:"id,omitempty"`
    Title   string    `protobuf:"bytes,2,opt,name=title,proto3" json:"title,omitempty"`
    Speaker *User     `protobuf:"bytes,3,opt,name=speaker" json:"speaker,omitempty"`
    Summary string    `protobuf:"bytes,4,opt,name=summary,proto3" json:"summary,omitempty"`
    Tags    []string  `protobuf:"bytes,5,rep,name=tags" json:"tags,omitempty"`
}
```

One message type



Multiple languages code

Less communication, More collaboration

with other platforms

1. Define message types
2. Generate source files
3. **Serialize/Deserialize**

Serialization

```
public func serializedData(partial: Bool = default) throws -> Data

public func jsonString() throws -> String

public func textFormatString() throws -> String
```

```swift
let user = User.with {
    $0.id = 1
    $0.type = .speaker
    $0.name = "kitasuke"
}

let talk = Talk.with {
    $0.id = 1
    $0.title = "Type-safe Web APIs with Protocol Buffers in Swift"
    $0.speaker = user
    $0.tags = ["swift", "iOS", "protocol-buffers", "altconf", "type-safe"]
}

let data = try? talk.serializedData()
```

Deserialization

```
public convenience init(serializedData data: Data,
    extensions: ExtensionMap = default,
    partial: Bool = default) throws

public convenience init(jsonString: String) throws

public convenience init(jsonUTF8Data: Data) throws

public convenience init(textFormatString: String,
    extensions: ExtensionMap? = default) throws
```

```
let talk = try? Talk(serializedData: data)

let title = talk?.title
let speaker = talk?.speaker
let tags = talk?.tags
```

How serialization works

Binary Encoding

Key factor

1. Field number
2. Wire type

Field number

```
message Talk {
    int32 id = 1; ← // Field number
    string title = 2;
    string desc = 3;
    User speaker = 4;
    repeated string tags = 5;
}
```

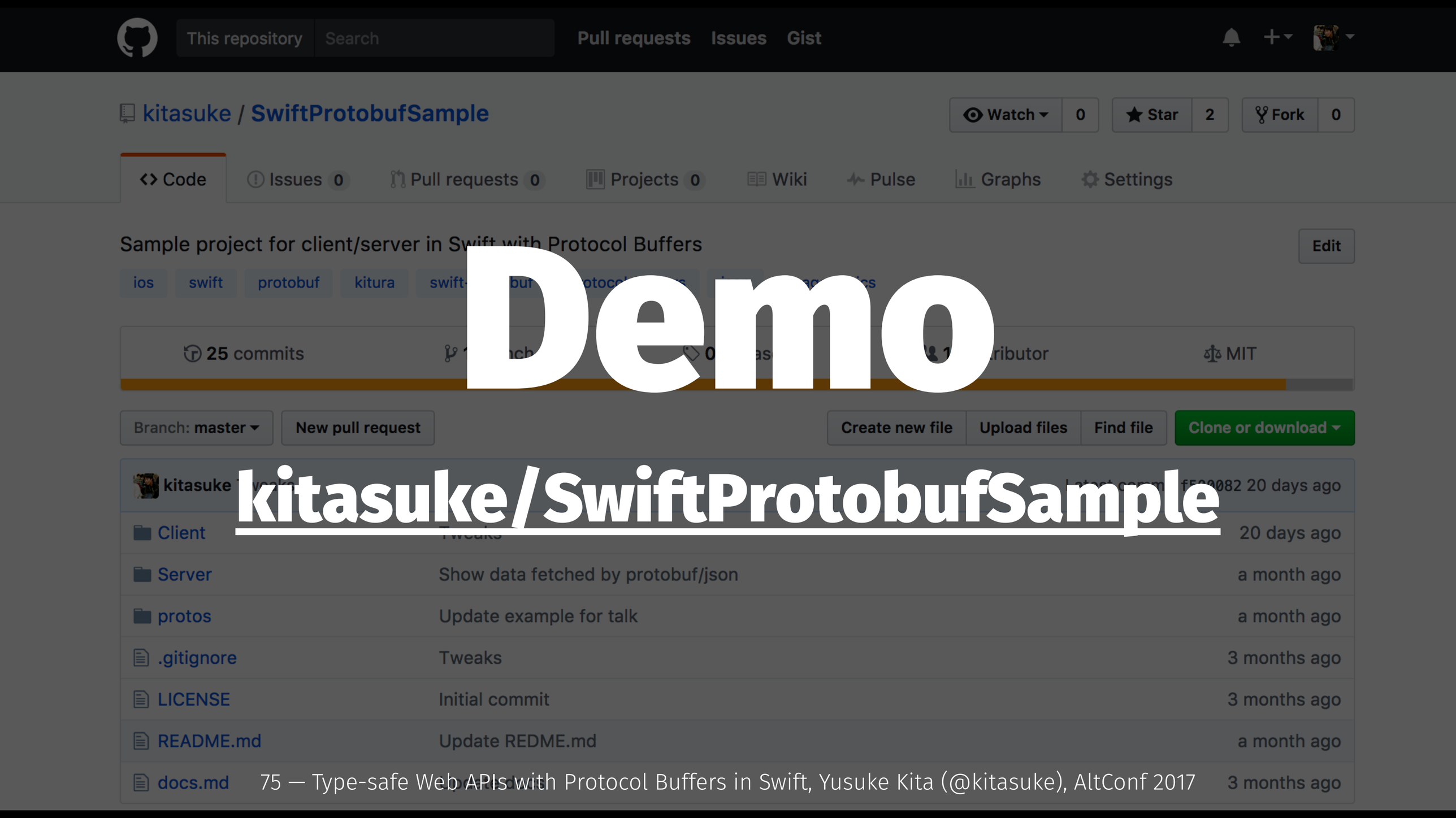
Wire type

Type	Meaning	Used For
0	Varint	int32, int64, uint32, uint64, sint32, sint64, bool, enum
1	64-bit	fixed64, sfixed64, double
2	Length-delimited	string, bytes, embedded messages, packed repeated fields
3		(deprecated)
4		(deprecated)
5	32-bit	fixed32, sfix3d32, float

```
// message type
message Test1 {
    int32 a = 1;
}
test1.a = 300

// encoded message
08 96 01


08 // field number and wire type
96 01 // value which is 300
```

Small and Numeric

High network performance


This repository
Search
Pull requests
Issues
Gist
kitasuke / SwiftProtobufSample
Watch 0
Star 2
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Pulse
Graphs
Settings
Sample project for client/server in Swift with Protocol Buffers
Edit
ios
swift
protobuf
kitura
25 commits
MIT
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
kitasuke
Client
Server
Show data fetched by protobuf/json
protos
Update example for talk
.gitignore
Tweaks
LICENSE
Initial commit
README.md
Update REDME.md
docs.md
20 days ago
a month ago
3 months ago
Demo
kitasuke/SwiftProtobufSample

Quick Recap

Quick Recap

→ Type-safety

Quick Recap

→ Type-safety

→ Shared data model

Quick Recap

→ Type-safety

→ Shared data model

→ High performance

You might have concerns about...

Versioning

Backward compatibility

→ Unknown field is ignored

→ Default value for missing field

Backward compatibility

→ Unknown field is ignored

→ Default value for missing field

as long as you don't change existing field number or wire type

Coexistence of protobuf & JSON

Absolutely you can!

Accept & Content-Type

→ `application/protobuf` - protobuf

→ `application/json` - JSON

HTTP Request

```
var request = URLRequest(url: url)
if protobuf {
    request.setValue("application/protobuf", forHTTPHeaderField: "Content-Type")
    request.setValue("application/protobuf", forHTTPHeaderField: "Accept")
} else if json {
    request.setValue("application/json", forHTTPHeaderField: "Content-Type")
    request.setValue("application/json", forHTTPHeaderField: "Accept")
}
request.body = try? userRequest.serializedData()
```

HTTP Response

```
URLSession.shared.dataTask(with: request) { (data, urlResponse, _) in
    let httpURLResponse = urlResponse as? HTTPURLResponse
    let contentType = httpURLResponse?.allHeaderFields["Content-Type"] as? String

    let response: Response?
    if contentType == "application/protobuf" {
        response = try? Response(serializedData: data!)
    } else if contentType == "application/json" {
        response = try? Response(jsonUTF8Data: data!)
    }
}
```

Sounds good so far 

So, what are Cons?

Not human-readable

→ Binary data is not understandable

Time investment

→ Time consuming at the beginning

→ Involvement from other platforms

Swift version

→ Watch Swift version of protobuf plugin

→ Specify tag version if you use older version

Stability

→ Still pre-release version only for Swift ⚠

→ Contribute if you find a bug 🚀

Where should we start?

Internal service

→ Easy to adapt

→ Small start

Data store

→ Database

→ NSKeyedArchiver

→ NSCache

Parsing library

→ JSON parser

→ Server side Swift

Conclusion

Conclusion

→ Swifty

Conclusion

→ Swifty

→ Consistent in Cross-platform

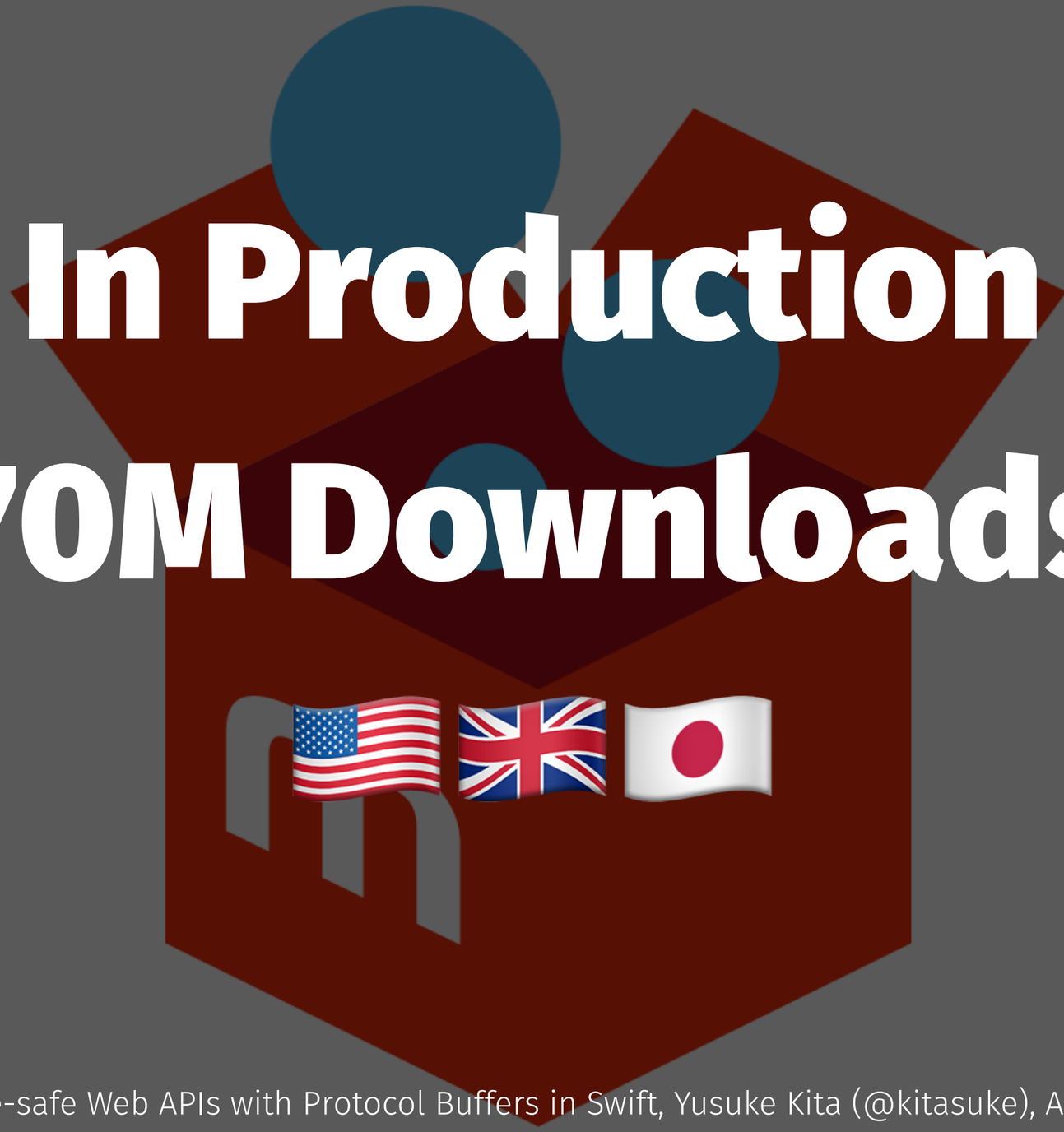
Conclusion

→ Swifty

→ Consistent in Cross-platform

→ Better performance

It's definitely worth it 💪


In Production
70M Downloads
🇺🇸 🇬🇧 🇯🇵

One more thing...

Codable 

```
/// A type that can convert itself into and out of an external representation.
public typealias Codable = Decodable & Encodable
```

```swift
/// A type that can encode itself to an external representation.
public protocol Encodable {

    /// Encodes this value into the given encoder.
    ///
    /// If the value fails to encode anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder to write data to.
    public func encode(to encoder: Encoder) throws
}
```

```swift
/// A type that can decode itself from an external representation.
public protocol Decodable {

    /// Creates a new instance by decoding from the given decoder.
    ///
    /// This initializer throws an error if reading from the decoder fails, or
    /// if the data read is corrupted or otherwise invalid.
    ///
    /// - Parameter decoder: The decoder to read data from.
    public init(from decoder: Decoder) throws
}
```

```
struct UserRequest: Codable {
    let id: Int
}

let encoder = JSONEncoder()
let userRequest = UserRequest(id: 1)
let data = try? encoder.encode(userRequest)
```

```
struct UserResponse: Codable {
    let user: User

    struct User: Codable {
        let name: String
    }
}

let decoder = JSONDecoder()
let response = try? decoder.decode(UserResponse.self, from: data)
let name = response?.user.name
```

```
struct NetworkError: Codable {
    let code: Code

    enum Code: Int, Codable {
    case unknown = 0
    case badRequest = 400
    case unauthorized = 401
    case forbidden = 403
    case notFound = 404
    case internalServerError = 500
    }
}

let decoder = JSONDecoder()
let response = try? decoder.decode(NetworkError.self, from: data)
let code = response?.code
```

```
Codable(
    isTyped: true,
    status: .excited
)
```

Awesome 

Credits

Protocol Buffers
swift-protobuf
Kitura
Protocol Buffers in your Kitura Apps

Thank you!

GitHub: https://github.com/kitasuke
Demo: SwiftProtobufSample
Twitter: @kitasuke
Email: yusuke2759@gmail.com